Problem No. 1: The Blocks Problem

The algorithm manipulates blocks by performing four different types of valid commands and ignores any illegal commands.

The algorithm starts by reading the number of blocks 'n' from the input. It initializes an integer array 'blocks' of size 25x25 to represent the state of the block world. Each row represents a block slot, and the first element of each row [0], represents the number of blocks in this slot. The other elements of the row represent the blocks stacked int the slot.

The algorithm then enters a loop that reads each command from the input as string until it encounters the "quit" command. For each line entered, it extracts the command, type of movement and the two block numbers a and b, then gets their location in the 'blocks' array using the 'find' function(slot number &floor).

If the command is illegal (a = b or a and b are in the same stack of blocks), the algorithm ignores the command and moves on to the next command.

If the command is valid, the algorithm performs the required action based on the command type. For the "move" command, it returns all the blocks above block 'a' to their initial positions by calling the “returnblocks” function. For the "onto" command, it returns all the blocks above block 'b' to their initial positions. For the "pile" command, it moves the pile of blocks; block 'a' and any blocks above it while retaining the order of the blocks above 'a'. For the "over" command, it leaves the blocks staked on top of block ‘b’ as they are and then put block/pile ‘a’ on top.

The algorithm updates the 'blocks' array after performing each command by moving the blocks from their old positions to their new positions based on the type of command. It also updates the number of blocks in the slots.

The “find” function: it takes the desired block number ‘i’ to find and the variables to store the location 'a' 'b'. It then searches through the 'blocks' array to find the location of block 'i'. It scans each row of the 'blocks' array to find the first occurrence of block 'i'. Once it finds the row that contains block 'i', it sets the value of 'a' to the row (slot) number and the value of 'b' to the column (floor) number where block 'i' is located. The function then returns.

The "returnblocks” function: The 'returnblocks' function takes as input the two location variables 'a' 'b'. It moves all the blocks stacked on top of the block at position 'a' starting from position 'b' back to their initial positions. For each block, it sets its row to the block's original position (i.e., the slot corresponding to the block's number) and sets the number of blocks stacked on that row to 1. It then sets the first element of the row to the block number. Finally, it updates the number of blocks stacked on block 'a' to be the position before the first block that was moved.

Finally, once all the commands have been processed, the algorithm prints the final state of the 'blocks' array to the output. For each block position numbered i ($0 \leq i < n$), the algorithm prints the slot followed by a list of blocks that appear stacked in that position with each block number separated from other block numbers by a space.

Problem No. 32: Recognising Good ISBNs

The algorithm is used to validate ISBN-10 codes. An ISBN-10 code is a unique identifier for books, consisting of 10 digits or 9 digits and a check digit. The check digit is calculated based on the first 9 digits using a specific formula. The algorithm checks whether a given ISBN-10 code is valid or not by computing the check digit and comparing it with the last digit of the code.

The algorithm reads ISBN-10 codes from the input until it reaches the end of the input. For each ISBN-10 code, it calls the 'isValidISBN' function to check if it is valid or not. The 'isValidISBN' function takes as input a string representing an ISBN-10 code and returns a boolean value indicating whether the code is valid or not.

The 'isValidISBN' function iterates over each character in the input string and performs the following operations:

1. If the character is a digit from 0 to 9, it adds its value to a running total and simultaneously adds the running total to a partial sum variable.

2. If the character is 'X' or 'x', it checks if it is the last character in the string. If it is, it adds 10 to the partial sum and the running total. Otherwise, it returns false since 'X' can only appear as the last character.

3. If the character is a hyphen '-', it ignores it and moves on to the next character.

4. If the character is not a digit, 'X', 'x', or '-', it returns false since the input string contains an invalid character.

After iterating over all the characters in the input string, the 'isValidISBN' function checks if the running total is divisible by 11. If it is, the ISBN-10 code is valid, and the function returns true. Otherwise, the ISBN-10 code is invalid, and the function returns false.

Problem No. 39: Horse Shoe Scoring

This algorithm works by taking the coordinates for each four throws of every turn and adding the appropriate score to the turns overall score. After the last turn in prints the turn number and score in a table.

First, we will initialize $R = 10$ (horseshoe radius) and $r = 1$ (post radius).

We input the A and B coordinates in a string, then extract them into 4 integer variables "x1" "y1" "x2" "y2". We calculate the distance between each point and the origins using $D = \sqrt{X^2 + Y^2}$ and store them so that $'Da'$ is the distance between 'a' and origins and $'Db'$ is the distance from 'b' to origins. It Keeps taking input from user until it encounters an empty line.

Then we test the three cases using relations between the two circles:

1)Ringer: First to see if the post is inside circle 'A' ; $Da < R + r$. But since it's only a semicircle we need to make sure it's in the right half, and for that we need to calculate the maximum distance between the center of the post and 'B' that is possible before it exits the half circle. That distance is $Db(\max) = \sqrt{(R + r)^2 + R^2}$. And so, the second condition for this case will be $Db < Db(\max)$.

2)Toucher: Here we can use the rule of intersection between two circles $R - r \leq Da \leq R + r$. And since it's a semicircle we must also add the $Db \leq Db(\max)$ condition (we added the equal in this case because Db can only equal $Db(\max)$ when the two circles are touching).

3)Swinger: Here we can use $Da > R + r$ to make sure the post is outside of the horseshoe. Then $Db < Db(\max)$ to make sure it's close enough that rotating about b with make the post intersect with the semicircle.

Each turn it calculates the score over 4 throws using the above conditions (ringer = 5 points, toucher = 2 points, swinger = 1 point). It then stores score in corresponding place in "score" array.

After it's done and all scores have been stored it prints them out in a table.

Group B Additional Problem 13:

The algorithm encrypts or decrypts a user specified text using two different methods: XOR and bit inversion. The program prompts the user to enter a text, choose an encryption or decryption method, and enter a key (if using XOR encryption). It then performs the selected operation on the text and prints the result. The program also gives the user the option to perform the inverse operation on the result.

The program uses two functions to perform the encryption/decryption operations: 'Xor' and 'Not'.

The 'Xor' function takes as input a string 'line' and an integer 'key'. It iterates over each character in the string and applies the XOR operation between the character and the key. The result is stored back into the string.

The 'Not' function takes as input a string 'line' and applies the bit-wise NOT operation on each character in the string. The result is stored back into the string.

The main program starts by reading a user-specified text from the input. It then prompts the user to choose between two encryption/decryption methods: XOR or bit inversion. If the user selects XOR encryption, the program prompts the user to enter a key. The program then calls the 'Xor' function or the 'Not' function based on the user's choice to perform the encryption or decryption operation on the input text.

The program prints the encrypted or decrypted text to the output and prompts the user to perform the inverse operation again by entering 1 or 0. If the user enters 1, the program performs the inverse operation on the result of the previous operation, and asks if you want to inverse again. If the user enters 0, the program exits.